

2011



USER GUIDE

Globalization and Localization

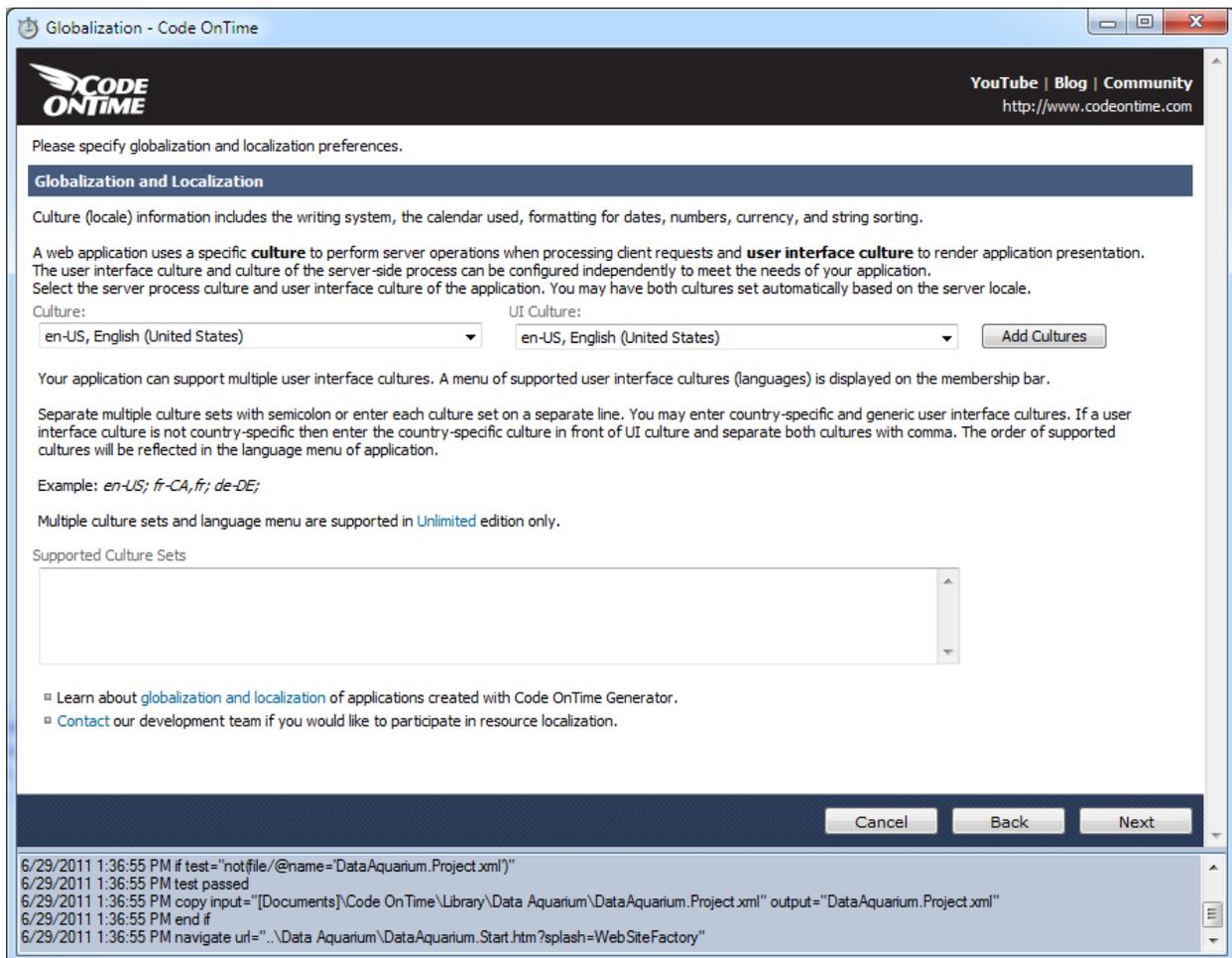
Globalization and Localization

Code On Time web application generator creates standard ASP.NET web projects and web sites, which take full advantage of the ASP.NET globalization infrastructure. The globalization support in ASP.NET has been perfected by Microsoft to allow creation of web applications that work with all cultures and languages. Our web application generator puts them together to offer a great solution for your globalization and localization needs.

Globalization

Code On Time applications provide full support for world cultures including date and time, calendar, numeric, and currency formats. Your application code may use one culture for the server side business logic and a different culture for presentation.

Globalization settings are configured on the *Globalization and Localization* page in the project wizard. The screen shot below shows this page in a new project.



Culture and *UI Culture* drop down lists are automatically configured to match the locale of your computer. The screenshot above shows both inputs set to *en-US, English (United States)*.

The selections in these drop downs are the primary cultures set in your application. *Culture* governs the culture used in the code executed on the server, while *UI Culture* controls the user interface culture aspects of the application.

If you are not planning to create applications for other locales then do not make any changes and simply click *Next* to continue project configuration. If you are developing an application for a locale that is different than the one selected by default in *Culture* and *UI Culture* then make sure to change the selections accordingly.

If you have the *Unlimited* edition of Code On Time Generator, then you have the ability to specify multiple supported culture sets. This can be done by selecting a combination using the *Culture / UI Culture* drop downs and pressing *Add Cultures* button. Do this for each culture set.

For example, if you are developing a line-of-business application that is expected to have users primarily in United States, you may have to account for users from the neighboring countries *Canada* and *Mexico* may need to be supported as well. In this age of global commerce it should not be a surprise that the business users of your web application may need to interact with partners from a distant country.

Let's use the example above. This application would need to support English, French, Spanish, and Traditional Chinese. The screen shot below shows configuration of the corresponding supported culture sets.

The screenshot shows a configuration window with two dropdown menus at the top: 'Culture:' and 'UI Culture:'. Both are set to 'zh-TW, Chinese (Taiwan)'. To the right of the 'UI Culture:' dropdown is a button labeled 'Add Cultures'. Below the dropdowns is a text area containing instructions: 'Your application can support multiple user interface cultures. A menu of supported user interface cultures (languages) is displayed on the membership bar. Separate multiple culture sets with semicolon or enter each culture set on a separate line. You may enter country-specific and generic user interface cultures. If a user interface culture is not country-specific then enter the country-specific culture in front of UI culture and separate both cultures with comma. The order of supported cultures will be reflected in the language menu of application. Example: en-US; fr-CA,fr; de-DE; Multiple culture sets and language menu are supported in Unlimited edition only.' Below the text area is a section titled 'Supported Culture Sets' with a list box containing the following items: 'en-US, en-US', 'es-MX, es-MX', 'en-CA, en-CA', 'fr-CA, fr-CA', and 'zh-TW, zh-TW|'. The list box has a vertical scrollbar on the right side.

You can also enter the support culture sets directly into the Culture Sets textbox as follows:

en-US, en-US; es-MX, es-MX; en-CA, en-CA; fr-CA, fr-CA; zh-TW, zh-TW;

Multiple culture sets are separated by semicolon or line breaks. *Culture* is separated from *UI Culture* by comma within each culture set definition. If both *Culture* and *UI Culture* are the same, you may enter just one culture. *UI Culture* may be non-specific and defined by two letters of the language (fr, es, en).

This particular example assumes that users in different locales do not share the server culture.

This is all the effort necessary to ensure that your application will correctly present and process date, time, calendars, numbers, and currency values in one or more locales.

Localization

All standard application resources will be automatically localized to the specified UI Culture languages. Our elegant localization system makes supporting multiple localized resources exceptionally simple and accessible.

Localization is one of the most complex aspects of application development. Various text fragments are typically dispersed in static application files such as pages, menus, reports, and the help system. Text messages are also emitted by business rules to report all sorts of errors and instructions to the end users of a web application.

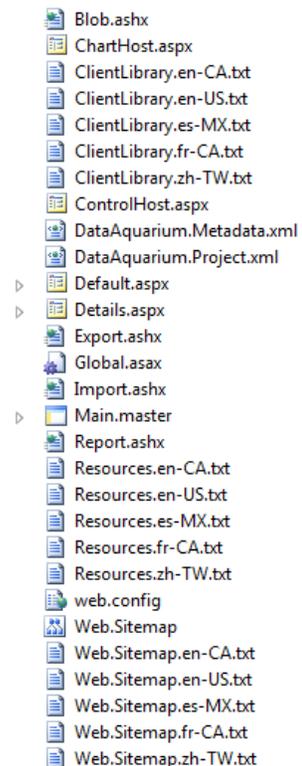
ASP.NET does provide standard means of externalizing application resources and creating localized versions of each resource file. A developer must maintain all resource files in sync and embed references to specific resources whenever a fragment of text needs to be referenced. In fact, a developer has to translate their entire application in the language of resource IDs that are subsequently translated into actual text written in the natural language. If your application is AJAX-based then you need to employ additional resource DLLs to ensure that localized text resources are available to the client scripts, which makes the localization process even more complex. Resource files in ASP.NET web applications have XML format, which requires users to exercise great care when changing them.

By using Code On Time Generator, this extensive task has become simple. Every generated Code On Time application includes several kinds of files commonly found in many hand-coded ASP.NET web applications:

- Ajax Client Library resources (*.js files)
- Data Controller Descriptors (*.xml files)
- Web Pages (*.aspx files)
- User Controls (*.ascx files)
- Site Maps (*.sitemap files)
- Core Library and Business Logic (*.cs or*.vb files)

We use a refreshingly simple and consistent method of localizing the application source code. If you open the generated application source code in *Visual Studio* or *Windows Explorer* then you will find a collection of text files that include the names of UI cultures supported in your web application.

The screen shot to the right shows the partial contents of the root folder of the example application discussed in the previous section. You can see ClientLibrary.*.txt, Resources.*.txt, and Web.Sitemap.*.txt groups of text files.



The first group defines the localized Client Library resources for all supported locales.

The second group defines localized resources used in the business rules and core library of the generated application.

The third group defines a collection of localized resources found in the *Web.Sitemap*, the file that describes the navigation hierarchy of the application.

You will find a few other clusters of localized resources if you browse the contents of the project. Notice that all of these clusters are associated with a specific static source file of your project much like *Web.Sitemap* and its satellite resources.

Let's take a look inside. Here are the first three lines from *ClientLibrary.en-US.txt*.

```
^About^About^About^  
^ActionBarActionsHeaderText^Actions^ActionBarActionsHeaderText^  
^ActionBarCancelActionHeaderText^Cancel^ActionBarCancelActionHeaderText^
```

Next example shows the first three lines from *ClientLibrary.fr-CA.txt*.

```
^About^À propos^About^  
^ActionBarActionsHeaderText^Actions^ActionBarActionsHeaderText^  
^ActionBarCancelActionHeaderText^Annuler^ActionBarCancelActionHeaderText^
```

This snippet shows the first three lines from *ClientLibrary.zh-TW.txt*.

```
^About^關於^About^  
^ActionBarActionsHeaderText^動作^ActionBarActionsHeaderText^  
^ActionBarCancelActionHeaderText^取消^ActionBarCancelActionHeaderText^
```

You have probably noticed the pattern that includes localization brackets on both sides of a localized resource. A localization bracket must start and end with “^” and may contain any combination of alphanumeric characters, such as ^About^, ^Label1^, ^23^. We call the combination of matching brackets and text between them a Localization Token.

This is an example of localization tokens *SiteHome*, *HomePath*, and *HomeDesc* found in *Web.Sitemap*.

```
<siteMapNode url="~/Default.aspx" title="^SiteHome^Home^SiteHome^" description="">  
  <siteMapNode title="^HomePath^Home^HomePath^"  
    description="^HomeDesc^Application home page^HomeDesc^"  
    url="~/Pages/Home.aspx" />
```

Sample code using localization token *RecordChangedByAnotherUser* is shown next. Method *Replace* of class *Localizer* automatically adds “^” to the localization token name and wraps it around the text fragment. This class is the core class of your application. You may find yourself using *Localizer.Replace* if you need to write custom business logic in a multi-lingual web application.

```
if (result.RowsAffected == 0)
{
    result.RowNotFound = true;
    result.Errors.Add(Localizer.Replace("RecordChangedByAnotherUser",
        "The record has been changed by another user."));
}
```

You can see that the localization token name is present along with the default text fragment in both use cases. This makes it much easier to understand the intended result. Your web application will remove the localization token at runtime and try to find the resource associated with the token in a file matched to the current web request UI culture. If the localized resource is found then it is used in place of the default value. If the exact match of a specific culture (*fr-CA*) is not found, then the localizer will try to find a file that matches the non-specific culture (*fr*).

All resource text fragments of the core server and client libraries are written in English. You can replace them if you change the localization files with the corresponding culture. For example, if your culture is *en-US* then change the files that end with **.en-US.txt* to replace the default English fragments. Do not change the corresponding source code files directly.

If you change the value between localization brackets and save the file then you have effectively changed the localized text representation of the corresponding physical resource of your web application. Simply run your application and observe your changes in action.

If your application is based on *Visual Studio* solution file, which is the case if you are developing a *Web App Factory*, *Azure Factory*, or *SharePoint Factory* project, then you will need to compile your application.

Files *ClientLibrary.*.txt* are not a part of your application. The web application generator will use the contents of these files to customize the *JavaScript* library of your application. If you change any definitions in *ClientLibrary.*.txt* file set then make sure to re-generate your project for the changes to take effect. Also make sure to hit *Refresh* button of your browser to ensure that the most recent version of localized resources is loaded in the web browser window.

Other localization file sets found in your project must be deployed along with the application.

Localization files found in the class libraries of your project (applies to *Web App Factory*, *Azure Factory*, *SharePoint Factory*) will have the culture component in the file name using “_” instead of “-”.

Here is a brief description of all standard localization files sets.

File Set	Folder	Description
ClientLibrary.*.txt	~/	JavaScript Client Library resources.
Resources.*.txt	~/	Resources used in the core application classes and business rules of application.
Web.Sitemap.*.txt	~/	Text and description of navigation nodes presented in application menu.
aspnet_Membership.xml.*.txt	~/Controllers	Membership manager resources.
aspnet_Roles.xml.*.txt	~/Controllers	Role manager resources.
TableOfContents.ascx.*.txt	~/Controls	Text fragments used in the standard table of contents that presents the site map as a tree.
Welcome.ascx.*.txt	~/Controls	Text fragments used in the welcome message.
Home.aspx.*.txt	~/Pages	Resources definitions found in the home page.
Membership.aspx.*.txt	~/Pages	Resource definitions found in the membership manager page.
Template.xslt.*.txt	~/Reports	Resource definitions used in RDLC report template.

Modified localization files are preserved by web application generator. If a new localization token is introduced in the core library and you have an existing application that does not have such a token then the code generator will insert the token in the file with a value equal to the default text fragment from the code generation library.

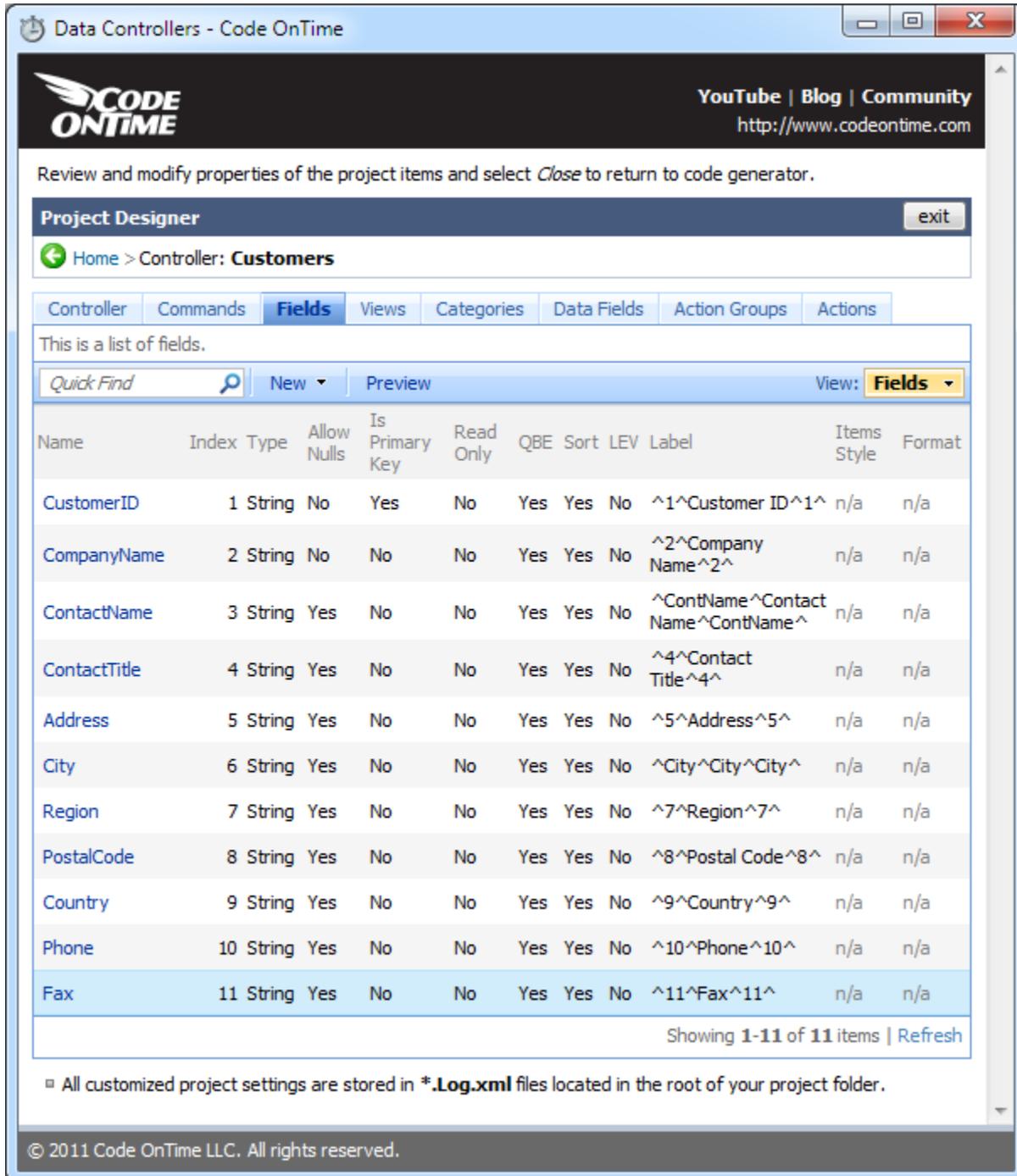
Multiple Cultures/Languages

Your application may support any number of user interface languages with the same code base. The application framework offers easy-to-use localization of static resources and simple API to render localized messages produced by your custom business rules. We have shown examples of localized static resources and business rules in the previous topic. This capability is only available in the *Unlimited* edition of Code On Time.

Automatic Translation

Wrap localization brackets around any text. Web application code generator will perform full translation in all languages supported by your application.

For example, the screen shot below shows the list of fields in *Customers* data controller. You can see that the field labels in the third column from the right have been changed to include localization brackets. Numerical and named tokens such as `^1^` and `^ContName^` are being used for localization.



The screenshot shows the 'Data Controllers - Code OnTime' application window. The 'Project Designer' is open for the 'Customers' controller, with the 'Fields' tab selected. The table below lists the fields and their properties, including localized labels.

Name	Index	Type	Allow Nulls	Is Primary Key	Read Only	QBE	Sort	LEV	Label	Items Style	Format
CustomerID	1	String	No	Yes	No	Yes	Yes	No	^1^Customer ID^1^	n/a	n/a
CompanyName	2	String	No	No	No	Yes	Yes	No	^2^Company Name^2^	n/a	n/a
ContactName	3	String	Yes	No	No	Yes	Yes	No	^ContName^Contact Name^ContName^	n/a	n/a
ContactTitle	4	String	Yes	No	No	Yes	Yes	No	^4^Contact Title^4^	n/a	n/a
Address	5	String	Yes	No	No	Yes	Yes	No	^5^Address^5^	n/a	n/a
City	6	String	Yes	No	No	Yes	Yes	No	^City^City^City^	n/a	n/a
Region	7	String	Yes	No	No	Yes	Yes	No	^7^Region^7^	n/a	n/a
PostalCode	8	String	Yes	No	No	Yes	Yes	No	^8^Postal Code^8^	n/a	n/a
Country	9	String	Yes	No	No	Yes	Yes	No	^9^Country^9^	n/a	n/a
Phone	10	String	Yes	No	No	Yes	Yes	No	^10^Phone^10^	n/a	n/a
Fax	11	String	Yes	No	No	Yes	Yes	No	^11^Fax^11^	n/a	n/a

Showing 1-11 of 11 items | Refresh

All customized project settings are stored in *.Log.xml files located in the root of your project folder.

© 2011 Code OnTime LLC. All rights reserved.

We have also changed various text properties of Customers page in designer.

Presentation

Page title is displayed in the title of the browser window.

Use symbol "]" in the page path to define a multi-level menu option that selects the page. Make sure that any segment of the path is matched to a path of an existing page that has an index less then the index of this page.

If *Path* is left blank then there will be no menu option to access the page.

Page description is displayed as a tool tip of the corresponding menu option.

Custom style is one or more CSS classes. Use *Wide* as custom style to eliminate the side bar on the page.

The page will feature *Aboutbox* on the side bar if specified.

Title *

Path

Description

Style *

Custom Style

About This Page

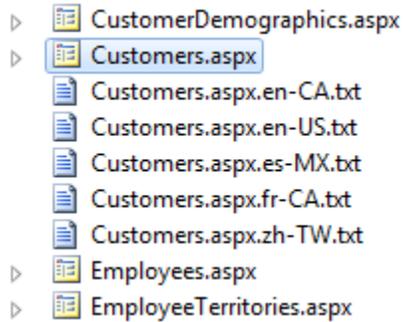
The following versions of *Customers* screen are presented if we generate our application and select languages corresponding to *fr-CA* and *zh-TW* cultures.

The image shows two side-by-side screenshots of a web browser displaying the 'Customers' page in different languages. The left screenshot is in French (fr-CA) and the right is in Chinese (zh-TW). Both show a table of customer data with columns for company name, contact name, title, address, city, region, postal code, country, and phone number. The French version has a sidebar with navigation links and a search bar. The Chinese version has a similar layout but with Chinese text and a different sidebar structure.

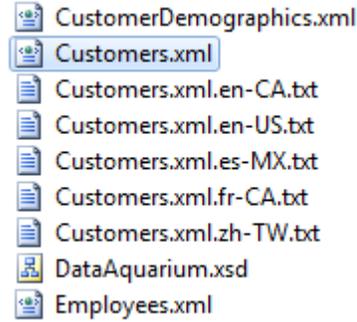
Nom de l'entreprise	Nom du contact	Titre du contact	Adresse	Ville	Région	Code postal	Pays	Téléphone
Alfredo Futterliste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	n/a			
Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.	n/a			
Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.	n/a			
Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	n/a			
Berglunds snabbköp	Christna Berglund	Order Administrator	Berguvsvägen 8	Luleå	n/a			
Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	n/a			
Blondessdél père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg	n/a			
Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid	n/a			
Bon app'	Laurence Labihan	Owner	12, rue des Bouchers	Marseille	n/a			
Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen BC	n/a			

If you inspect the generated application then you will notice the following new culture sets that were created by Code On Time web application generator.

Localization files for ~/Pages/Customers.aspx



Localization files for ~/Controllers/Customers.xml



Feel free to open any of these text files to refine the localized text fragments. Generated applications monitor localization files and will start using the fresh content when you save the changes.

Language Detection

Automatic detection of supported culture/language is a part of web applications generated with *Code On Time*. If the client browser culture is supported by your app then the appropriate localized resources are utilized to render the pages without any user involvement.

Web browsers send language preferences to web servers with each request. Culture manager of the generated web application will automatically match a supported culture set with the languages accepted by the user's browser. If a match is found then the culture set is automatically selected.

If the matching culture is not found then the application will use the default culture set of your application that was selected on the *Globalization and Localization* page of project wizard.

Language Selector

Membership bar offers a list of languages supported in your application with native names presented to application users. Language selection is automatically memorized and maintained with a sliding expiration.

Language selector complements automatic language detection.

We Need Your Help!

Automatic translation of localized resources is performed via *Google Translate*. The result of translation may not meet your expectations and we apologize for that.

We need your help with creating high quality localized standard resources. If you do make changes to any of the localized files listed below then please contribute your translations to benefit the developer community. You will find additional instructions in *ClientLibrary.*.txt* files in the root of your applications.

We are looking for help with the following files:

- ClientLibrary.*.txt
- Resources.*.txt
- Web.Sitemap.*.txt
- aspnet_Membership.xml.*.txt
- aspnet_Roles.xml.*.txt
- TableOfContents.ascx.*.txt
- Welcome.ascx.*.txt
- Home.aspx.*.txt
- Membership.aspx.*.txt
- Template.xslt.*.txt

Your contribution will be included in the general distribution of the code generation library. We will post the names of contributors on our blog with a link to the contributor's website if requested.